Combining Results of Accelerated Radiation Tests and Fault Injections to Predict the Error Rate of an Application Implemented in SRAM-based FPGAs

R. Velazco, G. Foucard, P. Peronnard, *Members, IEEE*

***Abstract*— An approach combining the SRAM-based FPGA static cross-section with the results of fault injection campaigns allows predicting the error rate of any implemented application. Experimental results issued from heavy ion tests are confronted to predictions in order to validate the proposed methodology.**

***Index Terms*— fault tolerance, fault injection, field programmable gate arrays, single event upsets, single event rates, accelerated testing, space applications,**

I. Introduction

FIELD Programmable Gate Arrays (FPGAs) are much appreciated by designers because they offer low cost, high performance, fast time to market and great flexibility for the design. Among the available technologies, SRAM-based FPGAs are well suited for space and avionic applications for their on-site reconfiguration [1]. Despite their attractive characteristics, designers are often reluctant to use SRAM-based FPGAs for critical applications, as the mission profiles may include harsh environments in which the device is exposed, for instance, to ionizing radiation [2][3]. Single-Event Effects (SEE) are a well-known threat for space systems, nowadays being also a concern for applications devoted to operate in the earth's atmosphere even at sea level.

The most probable consequences of impinging energetic particles on SRAM-based FPGAs are Single-Event Upsets (SEUs) and Multiple-Bit Upsets (MBUs) occurring either in the embedded design or in the configuration memory. Such faults occurring in the design are temporary and can be recovered by a Reset. However, faults induced in configuration memory are permanent and the only way to cope with them is to reconfigure the FPGA. Such faults may directly result in a "mutation" of the function implemented in the FPGA [4][5].

Manuscript received July 13th, 2010.

R. Velazco, Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble; e-mail : raoul.velazco@imag.fr

G. Foucard, Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble ; e-mail : gilles.foucard@imag.fr

P. Peronnard, Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble ; e-mail : paul.peronnard@imag.fr

Protection against SEUs in configuration memory must be taken into account by the designer [6]. Design level solutions, such as the well-known Triple Modular Redundancy (TMR) technique, are often adopted in order to build fault-tolerant architectures in SRAM-based FPGAs [7][8], but they always require finding a compromise between fault-tolerance and resource overhead or performance penalty.

Radiation accelerated tests are mandatory to obtain the sensitivity of the target device by determining its static cross-section curve. However, this figure significantly overestimates the sensitivity of a final application. In the case of complex circuits such as processors, previous researches [9] demonstrated that the dynamic cross-section of the final application can be accurately predicted combining the static cross-section with the results of fault injection campaigns in which the SEUs are emulated by a suitable approach (simulation, emulation, hardware/software fault injection….) whose choice depends mainly on the available circuit description.

The researches presented in this paper were done in the context of a participation of TIMA Laboratory to the LWS-SET (Living With a Star – Space Environment Testbeds) NASA program. The LWS project aims at studying the solar activity and the impact of its variations on the Earth and on life. The SET project is the element of LWS program that characterizes the space environment and its impact on integrated circuit and system reliability in space. The main goal of SET is to improve the engineering approach to accommodation and/or mitigation of the effects of solar variability on spacecraft design and operations. For this, the SET spacecraft provides several slots to embed experiments on the topic.

TIMA was in charge to develop an application able to observe the behavior of a commercial SRAM-based FPGA and its application. When the project started the most advanced device was selected as a test vehicle: a Xilinx Virtex-II 1000. In terms of logical capacity, this FPGA is a mid-range product among the Virtex-II family.

In this paper, a state-of-the-art approach devoted to predict the SEU error-rate will be applied to an application implemented on a SRAM-based FPGA. In section II the error

rate prediction method will be described. Fault injection campaigns were applied using an upgraded version of the THESIC (Testbed for Harsh Environment Studies on Integrated Circuits) platform developed at TIMA, so-called THESIC+. Details, experimental results and their confrontation with measures issued from heavy-ion test campaigns are summarized in section III. Finally, conclusions and perspectives are discussed in section IV.

II. Application Error Rate Prediction Method

The CEU (Code Emulated Upsets) methodology, presented for the first time in 2000 [9], is devoted to complex circuits such as processors and aims at determining a realistic prediction of the sensitivity of an application executed by the device under test (DUT).

A. General methodology

Let's suppose that a method allowing injecting bit-flips randomly in both the occurrence instant and target can be implemented for the considered DUT. The cross-section derived from the static test strategy provides the average number of particles of a given type which is necessary to provoke a bit flip of one of the memory cells included in the DUT.

If bit-blips are injected concurrently with the execution of a given program by software and/or hardware means, it can be derived an error rate, called t_{inj} in the following, as the number of detected errors divided by the number of injected bit flips:

$$\tau_{inj} = \frac{\#\,Errors}{\#\,Injected\ bit\ flips}$$

As t_{inj} can be interpreted as the average number of bit-flips needed to provoke an error in the executed program, the sensitivity to SEUs of the exercised program can be calculated by multiplying the underlying cross-section and the error rate issued from fault injection:

$$\tau_{SEU} = \sigma_{SEU} * \tau_{inj}$$

This approach to estimate the error rate was proposed and validated for processors. The main difficulty resides in the implementation of the fault injection strategy which must emulate as close as possible real bit-flips affecting the processor memory resources as the consequence of an SEU. The accuracy of derived error rates, when compared to the ones measured under radiation, strongly depends on the number of memory elements included in the studied circuit which are not accessible by means of the instruction set. Recent experiments performed using this approach [10], showed that predictions are very close from measured error-rates issued from radiation ground testing (heavy ion beams, protons) even for complex advanced processors, such as the Power PC 7448.

B. Porting the methodology to SRAM-based FPGAs

This methodology is very well suited for SRAM-based FPGAs as its two main limiting factors, the fault injection mechanism and the number of accessible bits through the instruction set, are not a concern. Indeed faults can be injected within the configuration bitstream (such a fault injection mechanism was presented in [11] and [12]), this does not require to modify the application. Moreover the whole configuration memory being accessible, any resource used by a design can thus be targeted. This particularity should improve the result accuracy of the methodology.

In the following are presented experimental results putting in evidence that the error rate of applications implemented in SRAM-based FPGA can accurately predicted by combining measured static-cross section with data issued from HW/SW fault-injection experiments performed off-beam during the execution of the final application.

III. Experimental Results

A. Testbed description

The test platform used in this work was the THESIC+ platform [13] with a daughterboard hosting the DUT: a Xilinx Virtex-II XC2V1000 FPGA [14]. Figure 1 depicts the architecture of the developed system, whose architecture is built around two FPGAs. The first one, called COM FPGA, contains a LEON2 processor. It handles the communication between the user's computer and the resources available on the THESIC's motherboard. It also monitors the DUT current in order to protect it against latchups. Data transfers are performed over an Ethernet network allowing a good data rate. The 2nd FPGA, called Chipset FPGA, contains the user design, mainly used to interface the DUT with the tester's resources.

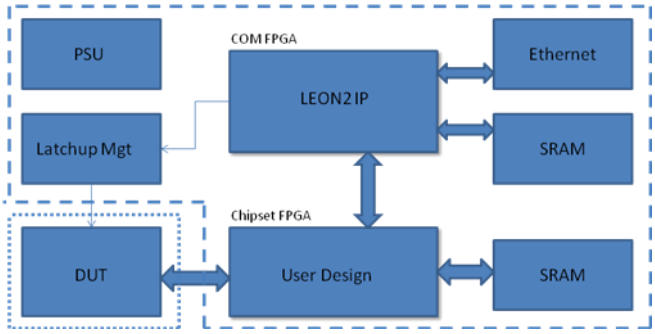

Figure 1 : THESIC+ block diagram

The FPGA application chosen for this study is based on a triple-DES (Data Encryption Standard) algorithm, also called DES3 [15]. The DES algorithm encrypts 64-bit data using a 56-bit key in 16 clock cycles whereas the DES3 encryption is achieved by three consecutive DES encryptions. Thus it requires three 56-bit keys and 48 clock cycles.

TMR architecture was implemented on the above described DES3 application. The current state of the TMR, the voter's output, is stored in a register which can be read by THESIC+.

When the voter does not detect an error, a zero value is loaded in the register. Otherwise the read value corresponds to the TMR's branch that produced results different from the ones of the two others replicas.

Having the possibility to observe the output of each TMR's branch would be a useful feature. However the experimental setup does not have the required number of inputs/outputs to implement that option. Thus a 3-bit register was added to the application in order to provide information on the behavior inside the chip. Register values are as follow:

- "0": all branches have the same value
- "1", "2" or "3": is the number of the branch which provides a result different from the two others.
- "4" : three different results
- "5", "6" and "7": N.A.

Moreover, the tester checks every result supplied by the TMR application in order to detect potential errors passing through the TMR comparator. This, combined with the content of the 3-bit register, allows distinguishing three types of errors:

- *Detected errors*: the 3-bit register detects an error on one of the branches, but the TMR is efficient and the output result is correct.
- *Falsely detected errors*: the 3-bit register reports a N.A. value and the application output is correct.
- *Undetected errors*: the 3-bit register reports no error, however the supplied application output is false.

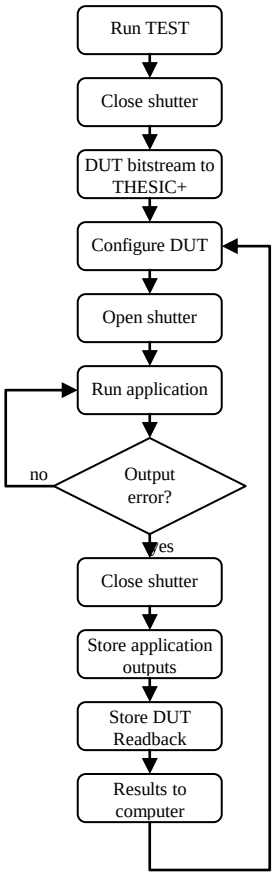

Figure 2 : Heavy ion campaign flow diagram

B. Radiation ground testing

Radiation ground testing was performed at the HIF (Heavy Ion Facility) cyclotron [16] of Louvain-la-Neuve (Belgium). Due to the allocated beam time, the studied TMR application was exposed only to two different particle species: Carbon and Argon. The test protocol is described in Figure 2.

Table 1 summarizes the number of application errors observed during a radiation ground testing session. These errors are classified according to the three categories described in § III.A.

Table 1 : Heavy ion accelerated test results

Ions	LET (MeV/mg/cm²)	Detected errors	Falsely detected errors	Undetected errors
Carbon	1.2	51	0	0
Argon	10.1	1,278	3	35

C. Fault injections

Fault injection campaigns where performed according to the above described methodology. The fault injection protocol is depicted in Figure 3. The total application execution length is 76 clock cycles. Faults are injected randomly during these 76 cycles.

In Table 2 are given results of a fault injection campaign in which the TMR application was executed 426,217 times. A fault was injected in each run at a random time and on a random target memory configuration bit. Thus, the average number of injected faults required to provoke each type of situation can be calculated by performing the ratio between number of observed faults and the total number of injected faults.

Table 2: Fault injection results

	Detected errors	Falsely detected errors	Undetected errors
Nb. of detected faults	14,564	237	319
Required average number of faults	3.42×10^{-2}	5.56×10^{-4}	7.48×10^{-4}

D. Observation of the effect of faults on memory bits configuring static and dynamic resources

Another concern was the impact of a permanent fault on the application. For example a fault can be injected after the resource is being used, thus not provoking an error on the application. However this error may still be present in the next application execution. This example takes place when the fault is injected on a memory cell configuring a static resource. On the other hand, a dynamic resource would be initialized upon application start, so a fault being injected before would not be harmful to the application.

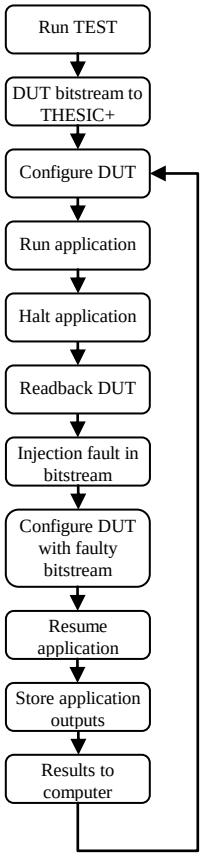

Figure 3 : Fault injection flow diagram

In order to observe this phenomenon the application was run twice for each injected fault. The fault is injected during the first run, then, the second run is done without modifying the application. 326,328 faults were injected during this experiment.

The first obtained results are presented in Table 3. They represent the number of injected faults that would produce an erroneous application result in one of the two runs or in both of them. The general observed tendency is that more errors are obtained during the second run than in the first run. This is a consequence of permanent faults on memory cells configuring static resources. Indeed a fault injected during the first run may not provoke an error during this run if the resource it configures is not used by the application during the remaining processing tasks. However the error will appear during the second run as the fault is still present in the configuration memory at very beginning of the execution.

A reverse tendency is observed for *falsely detected errors* as fewer errors are recorded during the second run. A reason could be the type of logic resources used to make this function. Indeed this type of error appears only when the register takes an unexpected value. This could be the case of an SEU occurring on the register itself or on the signal routing inside the FPGA. So, the concerned resource's type is mainly dynamic bits, thus the register value is reset when the second run starts and the injected fault is wiped out.

Table 3 : Number of errors during first and second application's runs

	Detected errors	**Falsely detected errors**	**Undetected errors**
1st run	11,237 (3.44 %)	178 (0.05 %)	235 (0.07 %)
2nd run	13,646 (4.18 %)	161 (0.05 %)	350 (0.11 %)

Results presented in the following focus on the injected faults that produce different results in the two runs. The figures given in Table 4 differentiate three types of events:

- The number of fault injections having no impact on the application output during the first run but generating an error in the second run. This is a typical case of a *permanent fault* occurring when the resource is not used anymore in the processing flow. However the fault is still present during the second run and it provokes an error on the application output.
- The number of fault injections producing a visible error in the first run but not in the second. This is a *temporary fault* as it disappears whenever the application is reset. This type of fault affects bits configuring dynamic resources.
- The number of fault injections affecting the result during the two application runs. This is another example of permanent faults where the error is present on the resource when it is critical for the application.

Temporary faults are less frequent than their permanent counterparts. This is not really surprising as dynamic bits represent a minority among the whole resources used by the application.

Table 4 : Errors when application results differs in the two runs

	Number of output errors
1st result correct, 2nd result false	2535 (0.77 %)
1st result false, 2nd result correct	401 (0.12 %)
Both results false	1890 (0.58 %)

The impact of the injection time parameter is discussed in the following. Figure 4 presents the total number of errors versus the injection-fault time instant. The general tendency observed is an increase over the time. Indeed the later the fault is injected, the highest is the probability to provoke an error on the application output.

Few errors are reported during the first 20 clock cycles as this period corresponds to the data and key loading in the design. The related amount of logical resources used is about 5% of the total amount used by the entire application. Most of the errors occured between the 20th clock cycle and the end where the encrypting process takes places involving 94% of the resources.

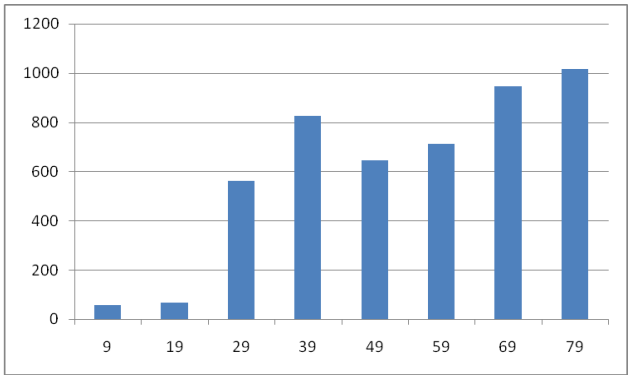

Figure 4 : Number of fault injections provoking an error in at least one run vs. the clock cycle during which the fault is injected

Details for each type of case enumerated in Table 4 are proposed in Figures [5-7]. Figure 5 summarizes the number of errors observed only during the first run versus the fault injection time. Those errors can be typically classified as temporary ones as they do not occur during the second run. Taking into account that the TMR weaknesses is the output comparator, direct hits on dynamic resources from the voter will most probably produce an undetected error.

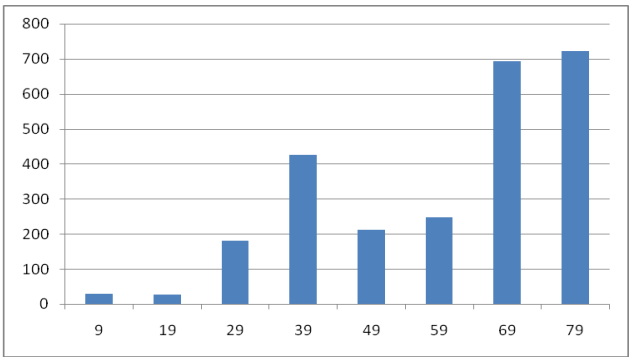

Figure 5 : Number of errors in the 1st run vs. instant of the fault injection when 2nd run is correct

Figure 6 illustrates the number of errors provoked on the application output versus the clock cycle during which the fault is injected when the 1st result is correct and the 2nd result is false. In this case, most of the errors take place during the encrypting period. As a fault injected early in the application execution has a highest probability to have an impact during the first run of the application, this result is clearly understandable.

Figure 7 depicts the results when both runs outputs false and different values. Only a few faults injected during the data loading period allowed observing this kind of behavior. This could be explained by the fact that logical resources are reused several times during the encryption process. Thus, a bit-flip corrupting the content of a static resource during this period will not produce a different result in the following runs during which the SEU is present from the beginning of the encryption.

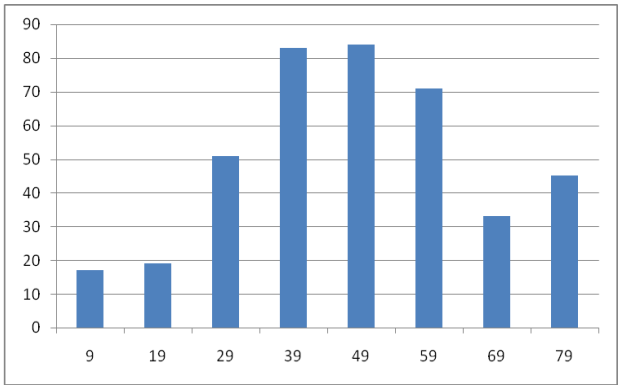

Figure 6 : Number of errors in the 1st run vs. instant of the fault injection when 2nd run is correct

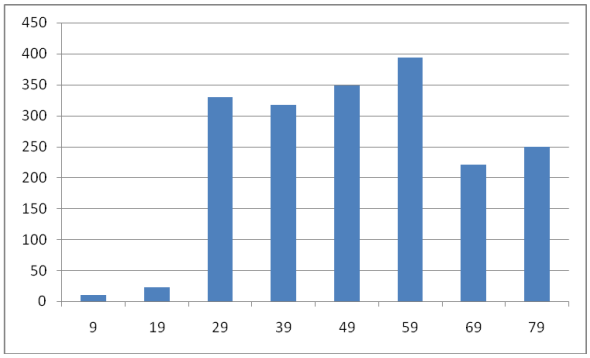

Figure 7 : Number of fault injections provoking different false results in both runs vs. the injection clock cycle

Figure 8 provides the results when outputs are identical and false in both runs. The tendency is to observe more errors when the fault injection takes place early in the application execution.

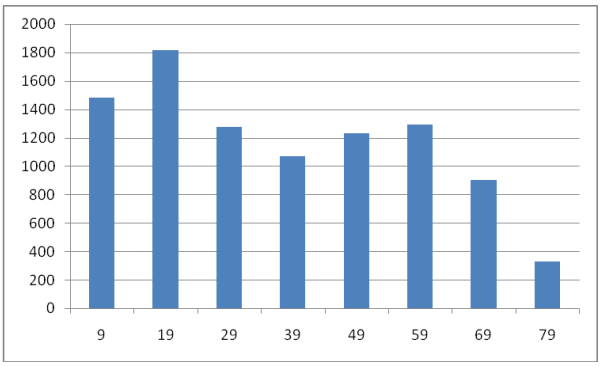

Figure 8 : Number of fault injections provoking identical false results in both runs vs. the injection clock cycle

In summary, the error profile as a function of the fault injection time will vary according to the type of errors observed and the nature of the logical resources being corrupted.

Table 5 : Measured vs. predicted error rates for the TMR implemented in the studied FPGA

Error rate	Particles	Detected errors	Falsely detected errors	Undetected errors
Measured	Carbon	$1.04x10^{-4}$	N/A	N/A
	Argon	$2.84x10^{-3}$	$6.67x10^{-6}$	$7.56x10^{-5}$
Predicted	Carbon	$9.53x10^{-5}$	$1.55x10^{-6}$	$2.09x10^{-6}$
	Argon	$1.94x10^{-3}$	$3.16x10^{-5}$	$4.25x10^{-5}$

E. Confrontation of measures and predicted results

The Carbon and Argon fluencies during heavy ion radiation ground testing were respectively 492,000 and 450,000. Thus, the application/device measured error rate can be obtained from the ratio between the number of detected errors and the particle fluency.

As described in §II.A the error rate of an application/device system can be predicted by multiplying the average number of faults required to provoke an error and the measured device SEU static cross-section. The measured static cross-section was $2.79x10^{-3}$ cm²/device for Carbon and $5.68x10^{-2}$ cm²/device for Argon.

Owing that no events of interest (neither falsely detected errors, nor undetected errors) were observed for Carbon, only the predictions done for Argon can be confronted to measures.

The analysis of these results, depicted in Table 5, shows that predictions are very close to measures: the maximum underestimation factor being less than 2. This difference might be the result of MBUs not being considered in the fault injection campaign.

Falsely detected errors follow a reverse trend: the prediction overestimates the measure by a factor close to 5. This could be explained by the small number of errors of this type observed during radiation ground testing. It is important to note that in such a case, the prediction is certainly closer to the reality than the measure.

IV. CONCLUSION

A state-of-the-art approach devoted to predict the error rate of a processor executing an application was adapted to deal with applications implemented in SRAM-based FPGAs. Results obtained for a case study, a TMR version of a cryptocore implemented in a Virtex II FPGA, confirmed the validity of the adopted methodology.

Future work will be focused on obtaining more results from radiation ground testing in order to confirm the observed trend.. The improvement of the fault injection methodology in order to allow a realistic simulation of MBUs, faults which challenge state-of-the-art fault tolerance techniques, will constitute one of the main goals of future researches in this area.

Confronting these predictions and these measures to those expected from the same application running on-board LWS/Satellite is the ultimate goal of this work. The satellite launch is expected for October 2012.

REFERENCES

[1] Michael Caffrey. “Space-based Reconfigurable Radio”, in Toomas P. Plaks and PeterM. Athanas, editors, Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), pages 49–53. CSREA Press, June 2002.

[2] E. Normand, “Single-Event Effects in Avionics”, IEEE Trans. Nucl. Sci., Vol. 43, n° 2, pp. 461-474, April 1966.

[3] T. Ma, P. Dressendorfer, “Ionizing Radiation Effects in MOS Devices and Circuits”, Wiley Eds., New York, 1989.

[4] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, M. Wirthlin, “SEU-induced persistent error propagation in FPGAs”, IEEE Trans. Nucl. Sci., Vol. 52, n° 6, pp. 2438-45, 2005.

[5] M. Wirthlin , E. Johnson , N. Rollins, M. Caffrey , P. Graham, “The Reliability of FPGA Circuit Designs in the Presence of Radiation Induced Configuration Upsets”, Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, p.133, April 09-11, 2003.

[6] F. Lima Kastensmidt, G. Neuberger, R. Fernandes Hentschke, L. Carro, R. Reis, “Designing Fault-Tolerant Techniques for SRAM-Based FPGAs”, IEEE Design & Test, v.21 n.6, p.552-562, November 2004.

[7] F.L. Kastensmidt, L. Sterpone, L. Carro, M.S. Reorda, “On the optimal design of triple modular redundancy logic for SRAM-based FPGAs”, Proc. Of Design, Automation and Test in Europe (DATE) 2005, vol. 2, pp. 1290-5, 2005.

[8] L. Sterpone, M. Sonza Reorda , M. Violante , F. Lima Kastensmidt , L. Carro, “Evaluating Different Solutions to Design Fault Tolerant Systems with SRAM-based FPGAs”, Journal of Electronic Testing: Theory and Applications, v.23 n.1, p.47-54, February 2007.

[9] Velazco R., Rezgui S., Ecoffet R., “Predicting Error Rate for Microprocessor-Based Digital Architectures through C.E.U. (Code Emulating Upsets) Injection”, IEEE Transaction of Nuclear Science, Vol. 47, No. 6, Dec. 2000, pp. 2405-2411

[10] P. Peronnard, R. Ecoffet, M. Pignol, D. Bellin, R. Velazco, Predicting the SEU Error Rate through Fault Injection for a Complex Microprocessor, 2008 IEEE International Symposium on Industrial Electronics, (ISIE' 2008, Cambridge,UK), 30 juin-2 juillet 2008.Xilinx, “Virtex-II Platform FPGAs: Complete Data Sheet”, http://www.xilinx.com, March 2005.

[11] Sterpone, L.; Violante, M.; , "A New Partial Reconfiguration-Based Fault-Injection System to Evaluate SEU Effects in SRAM-Based FPGAs," Nuclear Science, IEEE Transactions on , vol.54, no.4, pp.965-970, Aug. 2007

[12] Swift, G.M.; Rezgui, S.; George, J.; Carmichael, C.; Napier, M.; Maksymowicz, J.; Moore, J.; Lesea, A.; Koga, R.; Wrobel, T.F.; , "Dynamic testing of Xilinx Virtex-II field programmable gate array (FPGA) input/output blocks (IOBs)," Nuclear Science, IEEE Transactions on , vol.51, no.6, pp. 3469- 3474, Dec. 2004

[13] F. Faure, P. Peronnard, and R. Velazco, “Thesic+: A flexible system for see testing”, in Proc. of RADECS, 2002.

[14] Xilinx, “Virtex-II Platform FPGAs: Complete Data Sheet”, http://www.xilinx.com, March 2005.

[15] Opencores: http://www.opencores.org/project,des

[16] G. Berger, G. Ryckewaert, R. Harboe-Sorensen, “CYCLONE – A Multipurpose Heavy Ion, Proton and Neutron SEE Test Site”, RADECS Workshop, pp. 51-55, 1